Intelligent Systems
Assignment 2 Report

Gal Gantar, Domen Kastelic

January 2024

1 Introduction

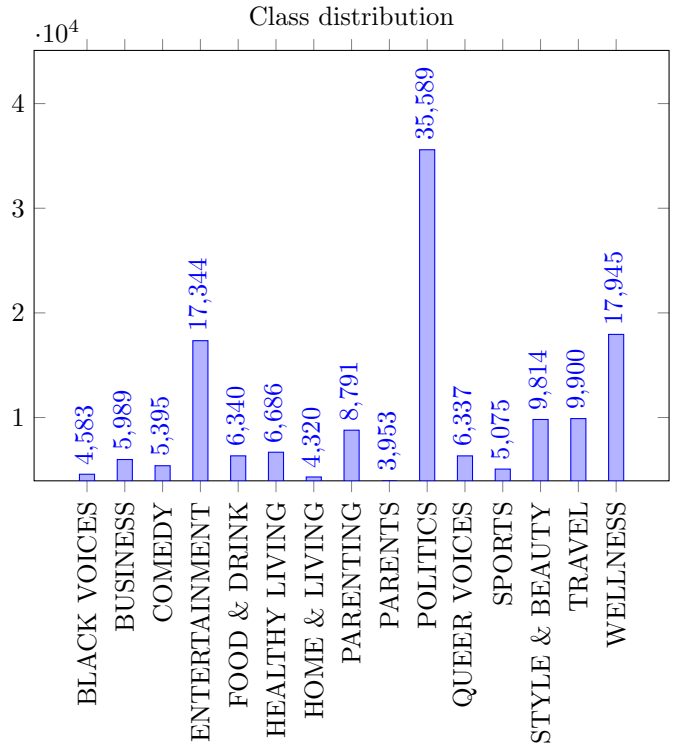
In this report we describe our solution to the second assignment in the Intelligent systems course. We decided early on to use a large language model as our main approach to article classification, but we also tried out some simpler models initially in order to have a baseline comparison besides the majority classifier.

We briefly explain our initial approach using logistic regression and random trees and continue with a more detailed explanation of our solution using an LLM. We go over how we pre-processed the data, generated embeddings using SBERT and how our text classification model is structured and optimised using hyper-parameter tuning.

Lastly, we present how all of the above-mentioned models performed and comment on the findings.

2 Initial approach

The provided dataset consisted of 148122 data points, 734 of which didn't have a headline, 12184 didn't have a short description and 61 were missing both. As our initial models were based on the headline and the description, we had to discard the data points that were missing both. That left us with 148061 data points for our initial models. This is when we looked at the distribution of the classes and thus the accuracy of the majority classifier as well:



We cleaned up the text by converting to lower-case, removing punctuation and stop words, lemmatizing and stemming words. For this we used functions provided by the NLTK library.

We trained a Word2Vec model on the processed data, with the input for each data point being the title and short description concatenated together. We used the vector obtained from averaging the vectors of all words from the title and the short description of a data point to represent that data point.

Then we split the data into a training and a testing subset, with 20% of the data going into the test set and used SKLearn's logistic regression and random forest models as initial models, to see how simple classifiers would perform when compared to the majority classifier and our second more sophisticated approach. The results are presented in a later chapter.

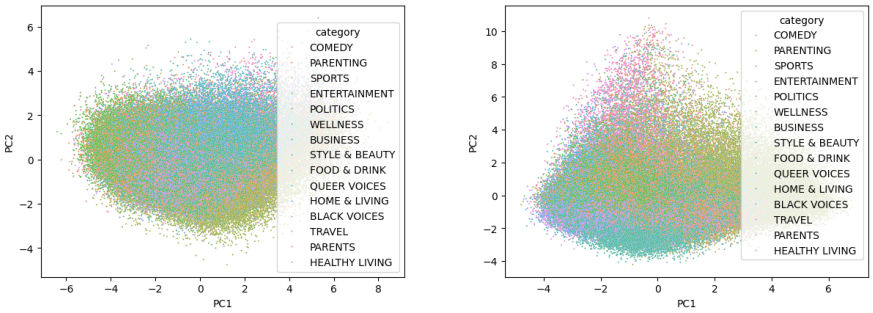
```
    # Logistic Regression model
    logistic_model = LogisticRegression()
    logistic_model.fit(X_train_w2v, y_train)
    logistic_predictions = logistic_model.predict(X_test_w2v)
    logistic_accuracy = accuracy_score(y_test, logistic_predictions
    )
    print("Logistic Regression Accuracy:", logistic_accuracy)

    # Random Forest model
    rf_model = RandomForestClassifier()
    rf_model.fit(X_train_w2v, y_train)
    rf_predictions = rf_model.predict(X_test_w2v)
    rf_accuracy = accuracy_score(y_test, rf_predictions)
    print("Random Forest Accuracy:", rf_accuracy)
```

3 Data preprocessing and embedding

For our second approach we decided to also utilize links to source websites that were included in the dataset. For every data point we parsed the website and downloaded its contents. Then we utilized the SBERT transformer model to produce two 768-dimensional numerical embeddings: the first embedding represented website text and the second was the embedding of concatenated headline and short description. After combining the two embeddings together we ended up with 1536 dimensional inputs for our model.

SBERT is a modification of google's BERT language model used to compute embeddings of sentences or even longer texts. Its main advantage over BERT is that it is faster while maintaining a similar accuracy [2]. It is implemented in the sentence_transformers python library, where you can also find many different pre-trained models. We used all-MiniLM-L6-v2 in particular.

We ran principle components analysis on both the earlier Word2Vec embeddings and the new SBERT embeddings. We reduced them to two dimensions to easily plot the data. There is a lot of overlap between categories, which suggests that two dimensions are not enough to reliably distinguish between them. But even in two dimensions, some differences between categories can still be observed.



(a) Word2Vec PCA

(b) SBERT PCA

We split the dataset into a global training set (90% of the entire dataset) and a global testing set (10% of the entire dataset). Before splitting we also shuffled data to prevent bias in our splits.

4 Details of Text Classification Model architecture

4.1 Model

For our architecture we chose a 3-layer neural network with dropout. The dimension of the first layer is between 256 and 1024 and the dimension of the second layer is fixed to 128. The first two layers increase model expresivity and the last layer acts as a prediction head, reducing the output dimension to 15 (for 15 possible classes). The predicted class is determined by the highest of all 15 components returned by the model. The exact model architecture depends on three hyper-parameters:

- **Dimension of the first layer** (ideally we would tune the dimensions of every layer) - larger dimensions increase the models expresivity but can also make data too complex for later layers to fully comprehend.
- **Weight decay** - regularization technique in machine learning that adds a penalty term to the loss function during training to encourage the model to have smaller weights.
- **Dropout probability** - the layer randomly sets a fraction of input (form 20% to 50%) units to zero during training, which helps prevent overfitting.

4.2 Optimizer

For the optimizer we chose the Adam optimizer. The exact optimizer architecture is determined by one hyper-parameter:

- **Learning rate** - the size of the steps that the optimization algorithm takes during the process of finding the minimum of the loss function.

4.3 Loss function

For our loss function we decided to use cross entropy loss, which is defined as:

$$CE = -\sum_{i=1}^{C} y_i \cdot \log(p_i)$$

The code that produces the model as well as defines search space for each of the hyper-parameters is defined below:

```
def create_model(hp):
    model = Sequential()
    model.add(Dense(units=hp.Int('units_1', min_value=256,
    max_value=1024, step=128),
                    activation='relu', kernel_regularizer=
    regularizers.l2(hp.Float('weight_decay', min_value=1e-6,
    max_value=1e-2, sampling='log'))))
    model.add(Dropout(rate=hp.Float('dropout', min_value=0.2,
    max_value=0.5, step=0.1)))
    model.add(Dense(units=128, activation='relu'))
    model.add(Dense(units=Y.shape[1], activation='softmax'))

    optimizer = Adam(learning_rate=hp.Float('learning_rate',
    min_value=1e-4, max_value=1e-2, sampling='log'))
    model.compile(optimizer=optimizer, loss='
    categorical_crossentropy', metrics=['accuracy'])
    return model
```

5 Preliminary hyper-parameter tuning

For our hyper-parameter tuning we further split global training set into local training set (80% of the global training set) and local validation set (20% of the global training set). Since it is possible (but very unlikely in our scenario since we don't have many hyper-parameters) for hyper-parameters to overfit we kept our global test set (10% of the entire dataset) out of the hyperparameter tuning process. Since weight decay and learning rate can occupy multiple orders of magnitude we used *log sampling* when tuning them.

We used the Hyperband tuner from the Keras tuner python library, which is a part of Tensorflow. Hyperband tuner uses an improved random search to find the best hyper-parameters. It relies on stopping the models that perform poorly early in order to free up resources and speed up the search [1].

After 28 rounds of Hyperband tuning the optimal hyper-parameters were:

units_1	896
weight_decay	4.3028e-05
dropout	3.0000e-01
learning_rate	4.0560e-04

Table 1: Best hyper-parameters

6 Model evaluation

After training the model on the training set we employed multiple metrics to evaluate the performance of our model on the testing set:

- **Accuracy:** The ratio of correctly predicted instances to the total instances, providing an overall measure of correct predictions.
- **Precision:** The proportion of true positive predictions among all positive predictions, indicating the accuracy of positive class predictions.
- **Recall:** The proportion of true positive predictions among all actual positive instances, measuring the model's ability to capture all positive instances.
- **F1 Score:** The harmonic mean of precision and recall, offering a balanced metric that considers both false positives and false negatives.

7 Validation

7.1 Cross validation

In order to more robustly evaluate our model we also employed cross-validation. After completing preliminary hyper-parameter tuning we split the global training set into three folds. For each fold we fist reset the model's weights, then trained the model for 10 epochs using the other two folds as a training set and the remaining fold as a testing set. At the end of each fold we then evaluated the model for that fold using the metrics described above. The final accuracy, precision, recall and F1 were computed as the averages of the measurements across all three folds.

```
for fold, (train_indices, test_indices) in enumerate(stratkf.split(
    X_train_global, np.argmax(Y_train_global, axis=1))):
    print(f"Training on fold {fold + 1}...")

    # Split the data for this fold
    X_train, X_test = X[train_indices], X[test_indices]
    Y_train, Y_test = Y[train_indices], Y[test_indices]

    # Train the model
    model = tf.keras.models.load_model("data/best_model")
    model.fit(X_train, Y_train, epochs=10, batch_size=32, verbose=
    True)

    # Make predictions on the test set
    predictions = model.predict(X_test)
    predicted_categories = np.argmax(predictions, axis=1)

    # Calculate metrics
    # ...

# Calculate and output the average metrics
average_accuracy = np.mean(accuracies)
average_precision = np.mean(precisions)
average_recall = np.mean(recalls)
average_f1 = np.mean(f1_scores)
```

7.2 Global testing

Finally, we trained our model yet again on the global training set for 10 epochs and evaluated all four metrics using the global testing set. This was done to ensure that our hyper-parameters did not overfit to the training data. The results obtained were in fact almost identical to the results achieved in cross-validation, which proves that hyper-parameters stayed general enough.

8 Results

8.1 Initial approach

The results of our initial approach using the majority classifier, logistic regression and random forest methods are presented here.

Model	Accuracy
Majority classifier	0.2404
Logistic regression	0.5533
Random Forest	0.4946

Table 2: Model accuracy

8.2 Cross validation and Global testing

The results for all three folds of cross validation and global testing on the remaining 10% of the data that was not used in hyperparameter tuning are presented here. As we can see, global training metrics are almost identical proving that the hyperparameters did not overfit.

Model	**Accuracy**	**Precision**	**Recall**	**F1**
Fold 1	0.7887	0.7885	0.7887	0.7872
Fold 2	0.7929	0.7946	0.7929	0.7889
Fold 3	0.7915	0.7905	0.7915	0.7897
Average	**0.7910**	**0.7912**	**0.7910**	**0.7886**
Global test	**0.7965**	**0.7963**	**0.7965**	**0.7899**

Table 3: Metrics for cross validation and global testing

8.3 Classification report

The classification report is comprised of precision, recall and F1 metrics computed for each of our classes. We can see that the classes with the lower metrics typically have small number of support points which means that the model did not observe enough instances of the selected class to learn how to distinguish it from the others. It might also be the case that some news classes have articles with similar internal structure (most common words, use of adjectives, style of writing) and other do not which makes them more difficult to classify.

Class	**Precision**	**Recall**	**F1**	**Support**
COMEDY	0.79	0.44	0.56	427
HOME & LIVING	0.67	0.64	0.65	565
BUSINESS	0.74	0.58	0.65	563
ENTERTAINMENT	0.82	0.84	0.83	1641
FOOD & DRINK	0.88	0.80	0.84	652
PARENTING	0.71	0.39	0.51	699
SPORTS	0.90	0.78	0.84	411
STYLE & BEAUTY	0.69	0.74	0.72	945
PARENTS	0.62	0.52	0.56	366
POLITICS	0.85	0.93	0.89	3554
HEALTHY LIVING	0.88	0.70	0.78	656
BLACK VOICES	0.84	0.80	0.82	521
BUSINESS	0.85	0.90	0.87	1001
TRAVEL	0.84	0.88	0.86	998
WELLNESS	0.70	0.86	0.77	1814

Table 4: Model Metrics per Class

9 Discussion

The results obtained from our model can be confidently proclaimed useful. The initial objective was to classify news articles and our model managed to do so successfully in roughly 80% of cases. Even the initial approaches using logistic regression and the random forest method with Word2Vec for text embedding have yielded promising results, and the performance improved even more when we started experimenting with the SBERT LLM and neural networks.

When performing hyper-parameter tuning the surprising result was that even though we were working with a relatively small 3-layer model the process of Hyperband tuning showed that the optimal results are obtained when using 896 neurons on the second layer instead of maximum 1024 neurons. Typically, more complex models produce better results (given that we have enough training data which in this case with 15 classes and 150k data points we do) however, the lower dimension might be a result of the next layer (128 neurons large) being to small to comprehend additional information.

The next surprising thing was the value of dropout during the hyper-parameter training. Dropout typically helps the model battle overfitting but in this case larger values of dropout seem to have lowered our model performance. During initial experimentation we also added another dropout layer between the second and the third layer but this made our model unable to learn anything.

The most obvious limitation of our model is its size. Since we were heavily limited by the hardware (all of the training was done on a laptop in less than 3 hours) we did not experiment with adding more layers of increasing the dimensionality. Another possible improvement would be to also include the second layer dimension in the hyperparameters instead of fixing it to 128. We could also experiment with different LLMs. Another possible improvement would be to try other types of models (Support vector machines, kNN, different NN architectures and so on). The final possible improvement would be to use statistical dimensionality reduction method such as UMAP (uniform manifold and approximation) to reduce SBERT or Word2Vec embeddings before feeding them to the neural network. This kind of approach would allow our model to only process the core components of larger vectors and that could remove the need for increasing model size and complexity.

In summary, while achieving a commendable 80% classification accuracy, potential improvements in model size, architecture, and the exploration of alternative techniques remain avenues for future enhancement.

References

[1] Lisha Li et al. *Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization*. 2018. arXiv: `1603.06560 [cs.LG]`.

[2] Nils Reimers and Iryna Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Nov. 2019. URL: https://arxiv.org/abs/1908.10084.